

Fig. 2

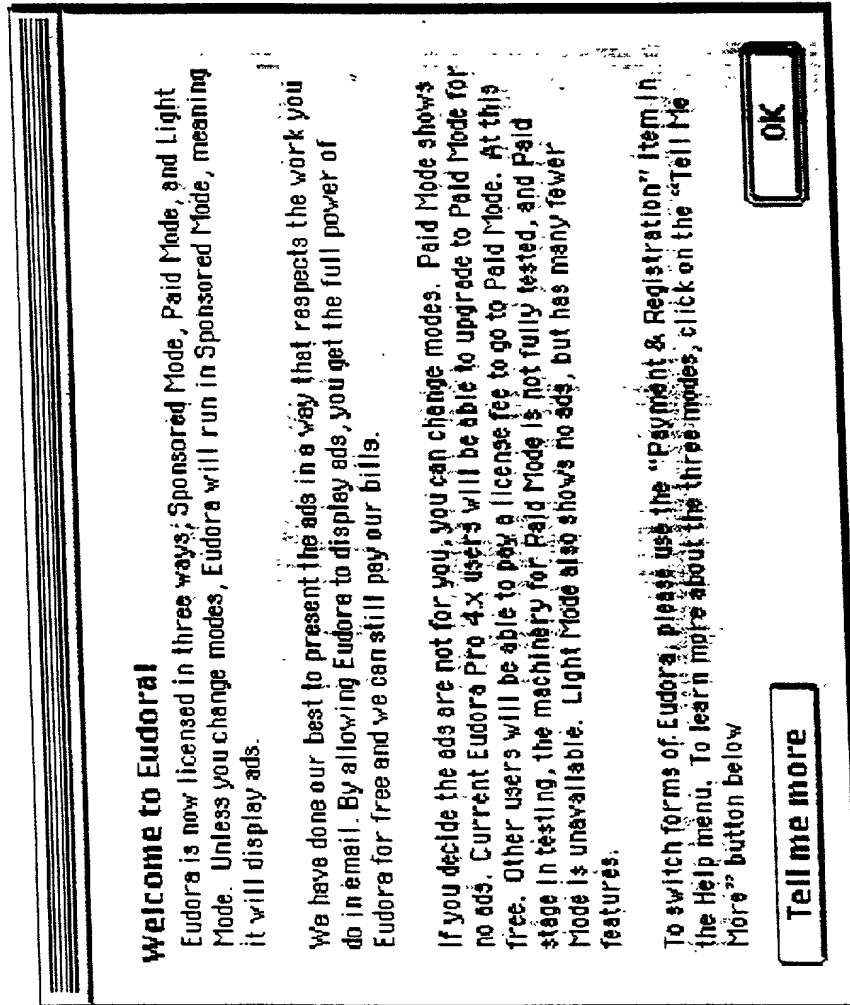


Fig. 4B

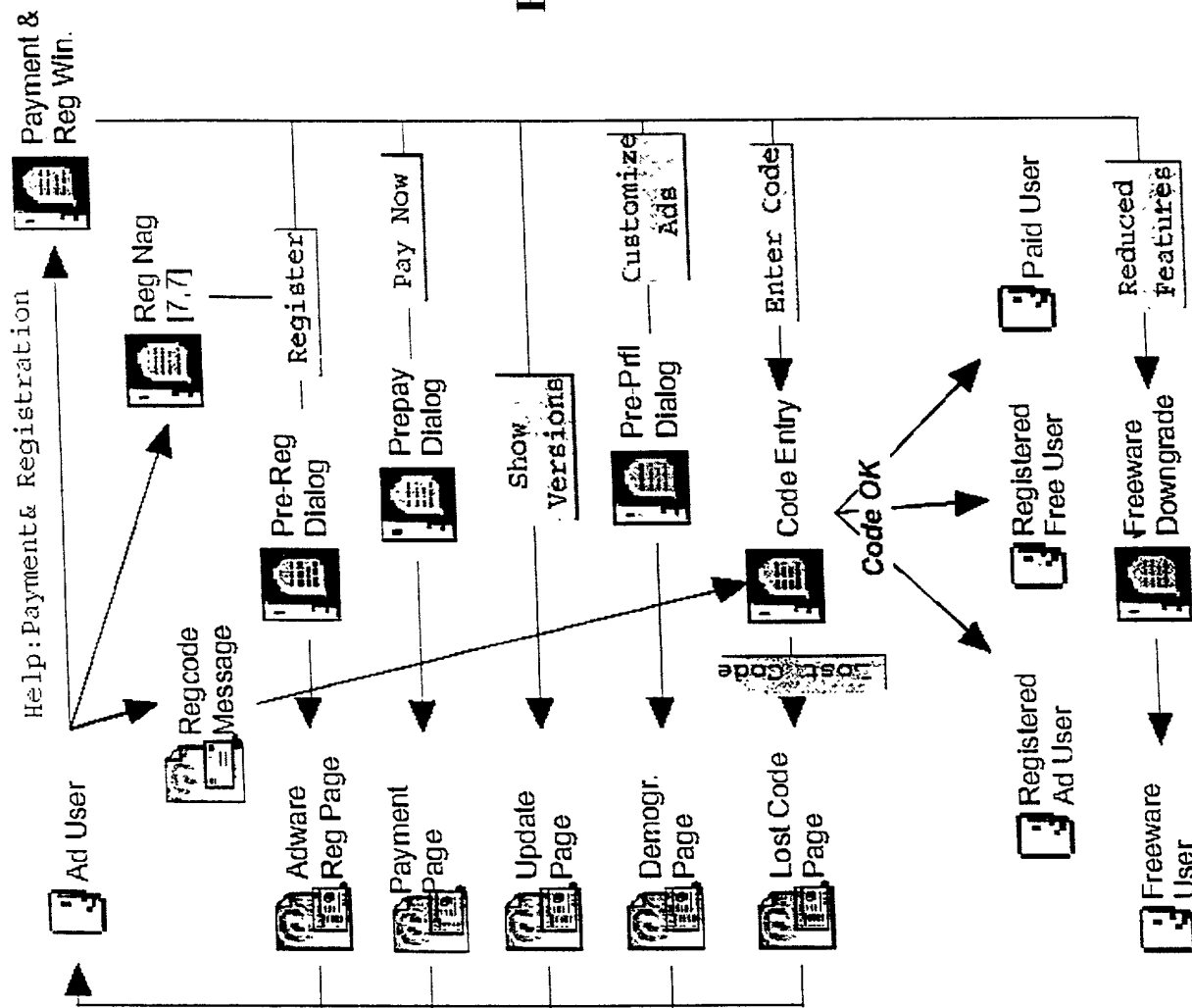


Fig. 5A








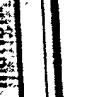
Payment & Registration		
Which Eudora is right for you?		
 Sponsored Mode (free, with ads)	 Paid Mode (costs money, no ads)	 Light Mode (free, fewer features)
Keeping Current		
 Register with Us	 Customize the Ads You See	 Find the Latest Update to Eudora
Your Registration Information		
<input type="text" value="no registration name"/>		 Change Your Registration
<input type="text" value="no registration code"/>		
 Take me to the Eudora website for more information		

Fig. 5B

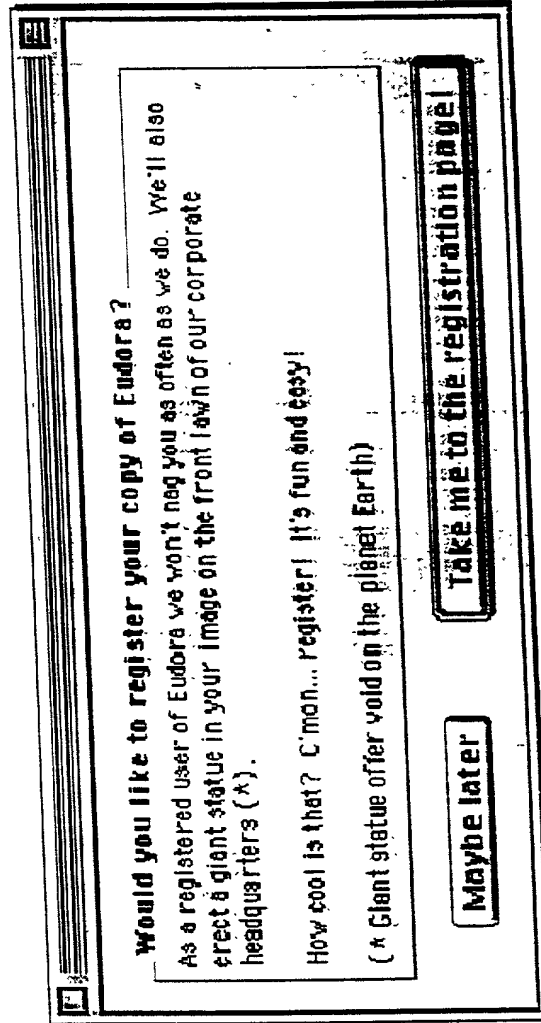


Fig. 5C

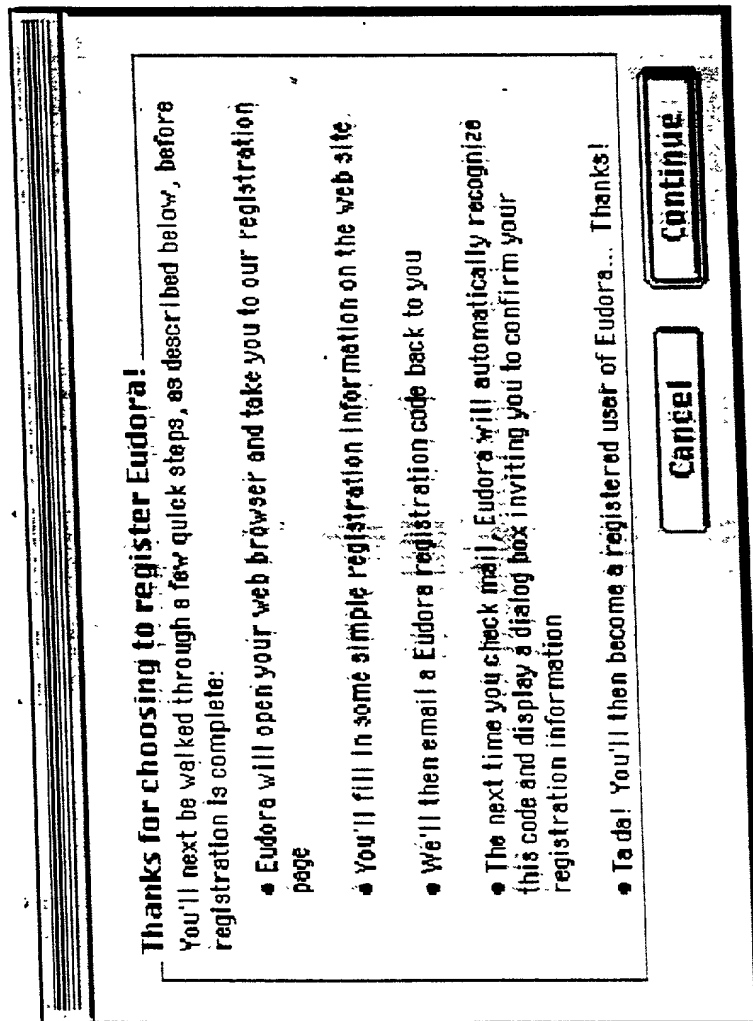


Fig. 5D

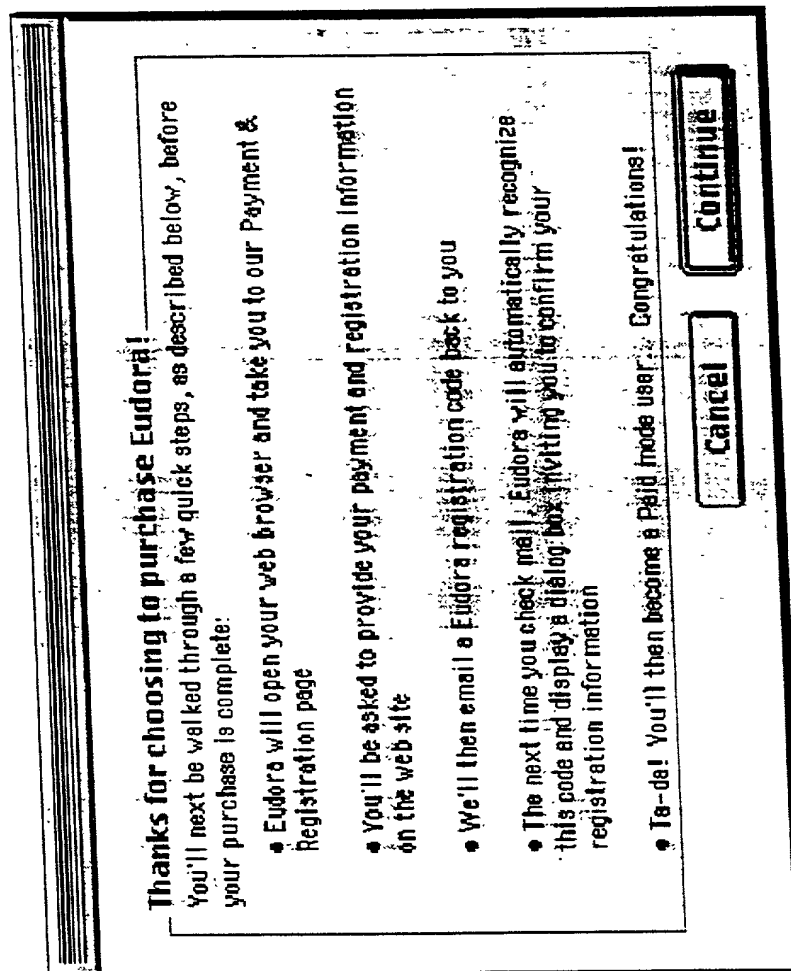


Fig. 5E

Thank you for your registration!

To complete your registration, please enter the name you
under and your registration code below.

The exact name you registered under:

First Name:

John

Last Name:

Manyjars

Your registration code:

48925-89A2-B1149

I lost the Code

Cancel

OK

Fig. 5F

There are updates available to Eudora

You have Eudora version 4.1. The following updates have become available since this version was released. If you'd like more information any of these updates, simply follow the links. If you'd rather not you of updates, follow this.

Eudora 5.0
This is a major upgrade, with great new features like automatic

Eudora 4.2
This update is mostly bug fixes. This update is free to you.

Printed Manual
You can buy a printed manual for Eudora.

Fig. 7B

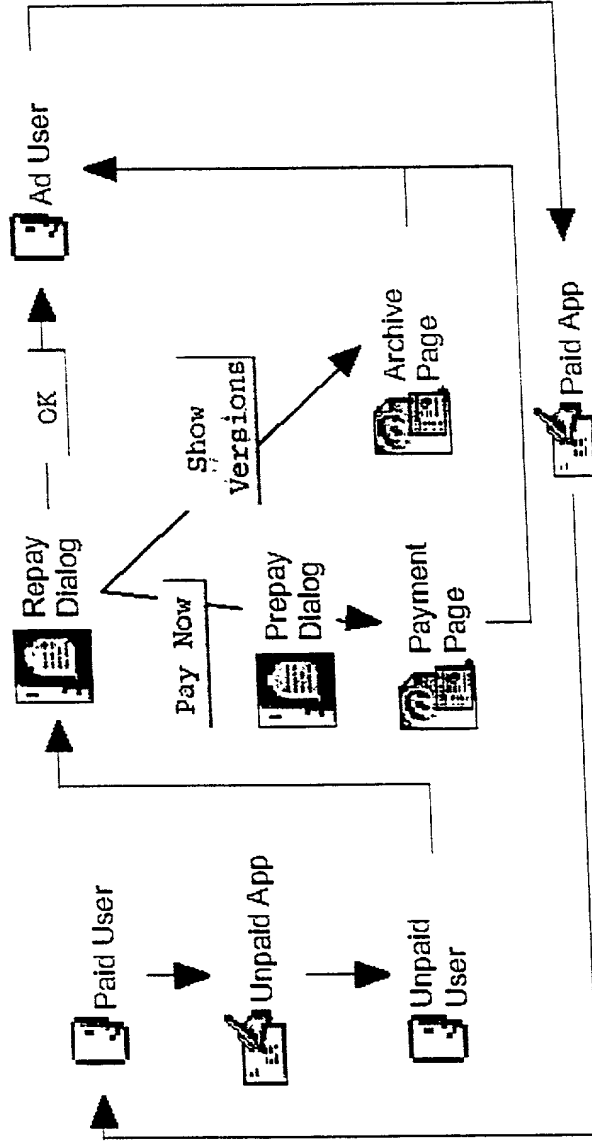


Fig. 9

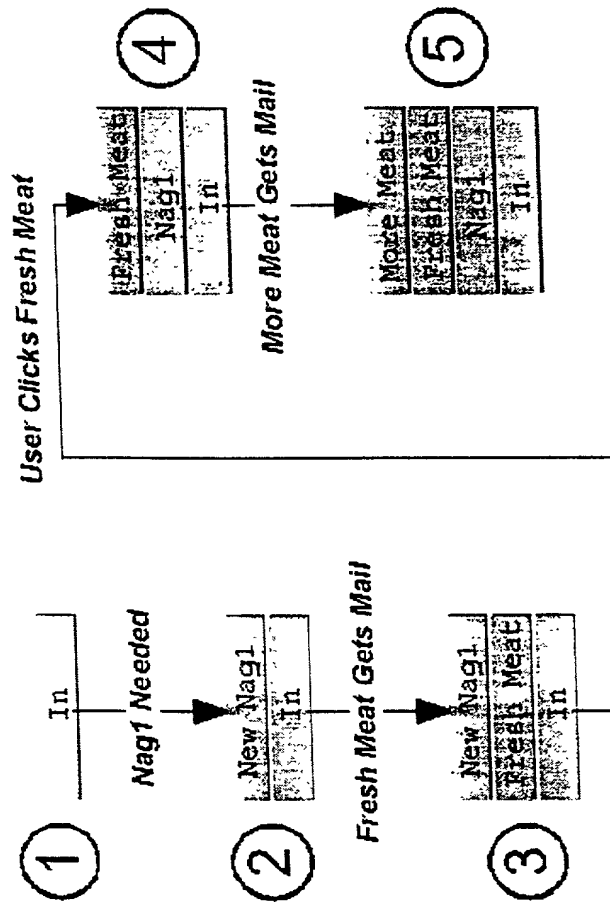


Fig. 10

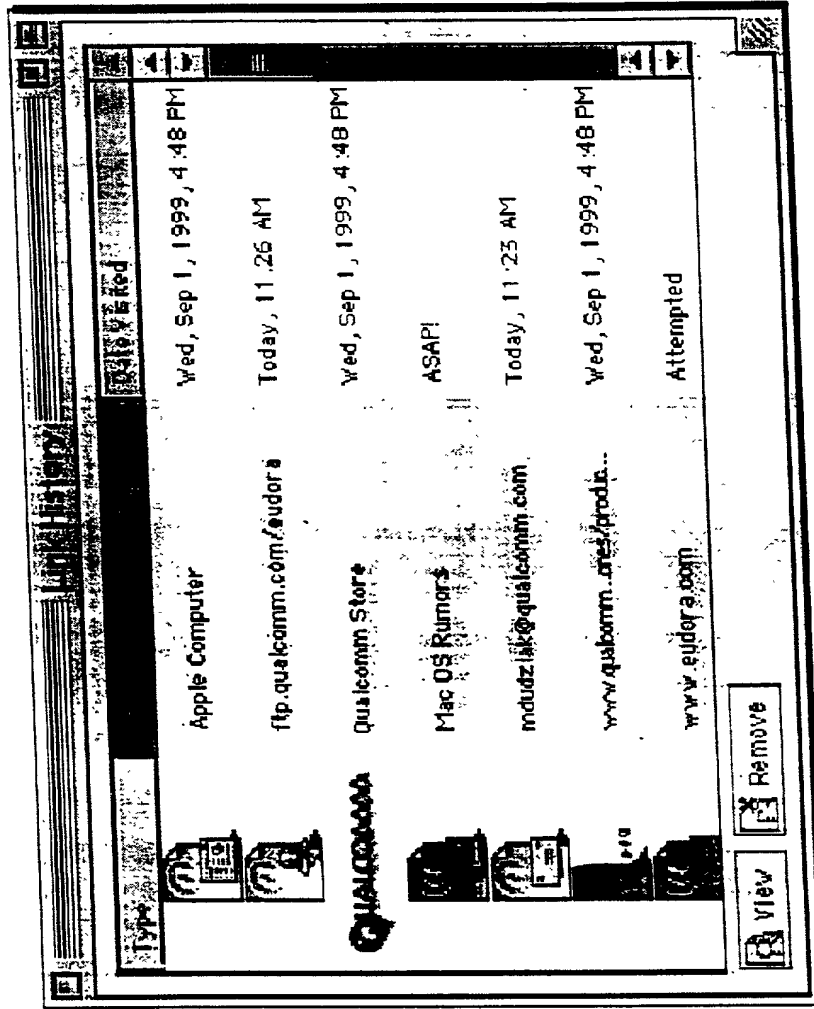


Fig. 12A

Performance	
Average Comm. Speed, Kbps	23.8
Average RJ Size, Kbytes	9.2
Number of Users	8,000,000
Number of Hours Running Today	2
Number Mailboxes Per User Per Hour	2
Playline Entry Size, Bytes	500

Fig. 13A

Implementation	
Day	3X Users
10	26
15	39
20	52
25	65
30	78
35	90
Average Comm. Speed, Kbps	
Average RJ Size, Kbytes	
Number of Users	
Number of Hours Running Today	
Number Mailboxes Per User Per Hour	
Playline Entry Size, Bytes	
3X Users Playline- Nbrs /	
RJ RJ Nbrs / RJ RJ Size Playline- Nbrs /	
User PC Download RJ RJ 100,000 Comm. Speed 100,000	
Check, Nbrs User N, 1000, Nbrs User	
10	2.4
15	3.6
20	4.8
25	6.0
30	7.2
35	8.4
0.1	
0.1	
0.1	
0.1	
0.1	
0.2	

Fig. 13B

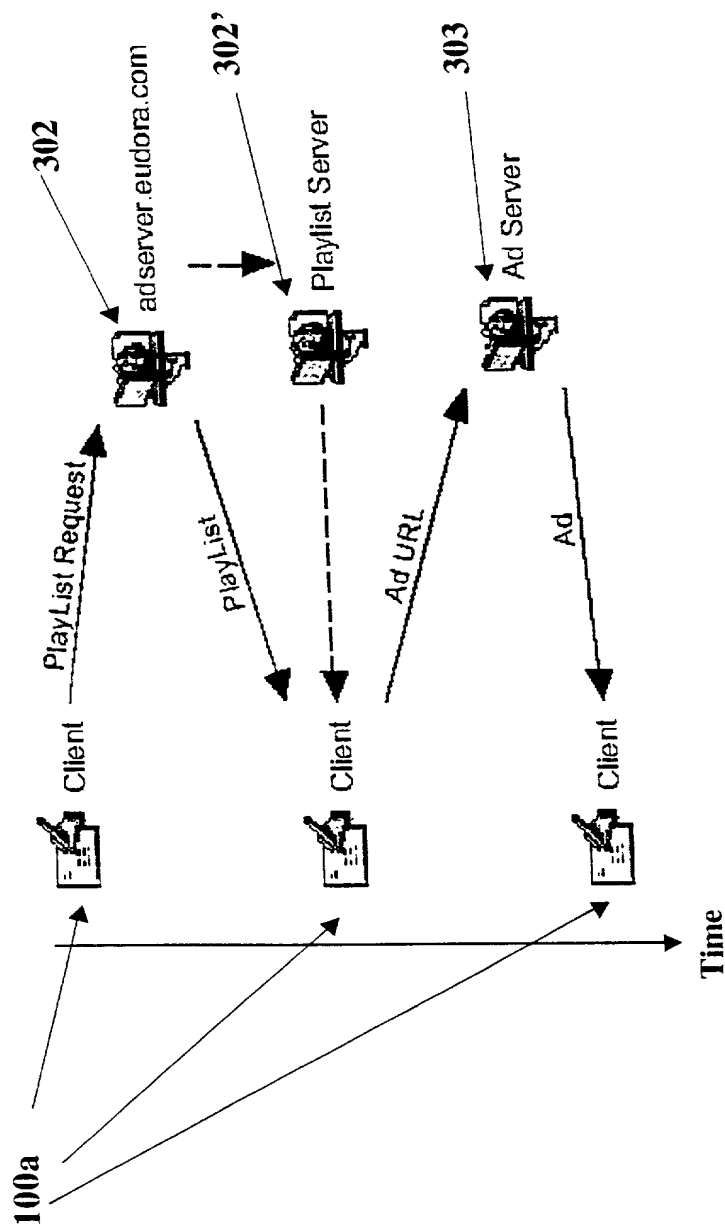


Fig. 14

```

////////////////////////////////////
// Main ad scheduler
ScheduleMain
{
// Has a new day dawned?
Do CheckForNewDay
// Are we are within the current ad's showFor?
if ( ad.thisShowTime < ad.showFor )
{
// there is nothing to be done
return
}
// At this point, we know that we need a new ad
// Perform housekeeping tasks on the old one
Do AdEndBookkeeping
// Pop out of a block if all ads on par
if ( block isn't all playlists )
{
find ad with minimum ad.numberShown
if ( ad.numberShown >= blockGoal )
set block to all playlists
}
// If we are over our quota of regular ads for the day,
// look for a runout
if ( adFaceTimeToday > faceTimeQuota )
{
Do ShowARunout
}
else
{
Do ShowARegularAd
}
}
// end ad schedule main

```

Fig. 15A


```

////////////////////////////////////
// We must perform certain tasks when the calendar day
changes.
CheckForNewDay
{if ( the calendar day has changed )
{
// Perform housekeeping tasks on the ad currently showing
Do StopShowingCurrentAd
// Runout ads are charged for a full showFor if they've been
shown
// at all on a given day. Charge any runout ads if they've
been
// shown at all.
for runout ads
{
if ( ad.thisShowTime > 0 )
{
ad.totalTimeShown += ad.showFor
ad.thisShowTime = 0
}
}
// Now, reset the counters for all ads to reflect the fact
that
// a new day has dawned.
for all ads
{
ad.numberShownToday = 0
}
// Record yesterday's facetime
// Might not literally be yesterday, be sure to use
// whatever day the app was last run on
set old current day's facetime to totalFaceTimeToday
// and reset our global regular ad facetime counter
adFaceTimeToday = 0
totalFaceTimeToday = 0
// if we were in a block, back out
set block to all playlists
}
}
// end CheckForNewDay

```

Fig. 15B

```

////////////////////////////////////
// This function shows a runout ad, and if it
// can't find one, goes to a rerun
ShowARunout
{
for runout ads
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this runout today?
if ( ad.numbersShownToday > ad.dayMax )
try next ad // this one's used up for the day
// are we done showing this runout for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next runout ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next runout ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this runout
// we are now in runout state
Do ShowAnAd
return
}
// if we haven't found a runout ad, we will go to "rerun"
state
Do ShowARerun
}
// end ShowARunout

```

Fig. 15C

```

////////////////////////////////////
// Rerun state. Look for a regular ad to rerun
ShowARerun
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// is this ad recent enough to rerun?
if ( ad.lastShownDate is older than returnInterval )
try next ad
// this one is too old to rerun
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, at this point we can show this ad, but because
// we're in rerun, we don't keep the books
Do ShowAnAd
return
}
// if we get here, we have no ads to show. Punt.
return
}
// end ShowARerun

```

Fig. 15D

```

////////////////////////////////////
// Show a regular ad
ShowARegularAd
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this ad today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we done showing this ad for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this ad
// we are now in regular state
Do ShowAnAd
return
}
// If we get here, we have failed to find a regular
// ad. Go to runout
Do ShowARunout
}
// end ShowARegularAd

```

Fig. 15E

```

////////////////////////////////////
// Perform necessary housekeeping when we're taking
// down an ad
AdEndBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// Account for at most ad.showFor seconds, provided
// we've shown the ad for at least ad.showFor seconds
// Note that this means we don't charge for time beyond
// ad.showFor seconds, which is important
if ( ad.thisShowTime >= ad.showFor )
{
ad.numberShownToday += ad.showFor
ad.shownFor++
// we do NOT reset thisShowTime here, we do it in
// AdStartBookkeeping. It actually doesn't matter where
// we do it, provided we are careful NOT to do it for
// runout ads.
}
}
// end AdEndBookkeeping

```

Fig. 15F

```

////////////////////////////////////
// Show an ad, including bookkeeping and block handling
ShowAnAd
{
// If the ad is in a block, notice that
if ( it's in a "block" playlist )
{
if ( not currently in a block )
{
find ad in block with minimum numbersShown
make that our ad
set blockGoal to minimum numbersShown+1
}
set current block to this playlist
}
// now do bookkeeping
Do AdStartBookkeeping
// and actually show it
Do DisplayThatAd
}

```

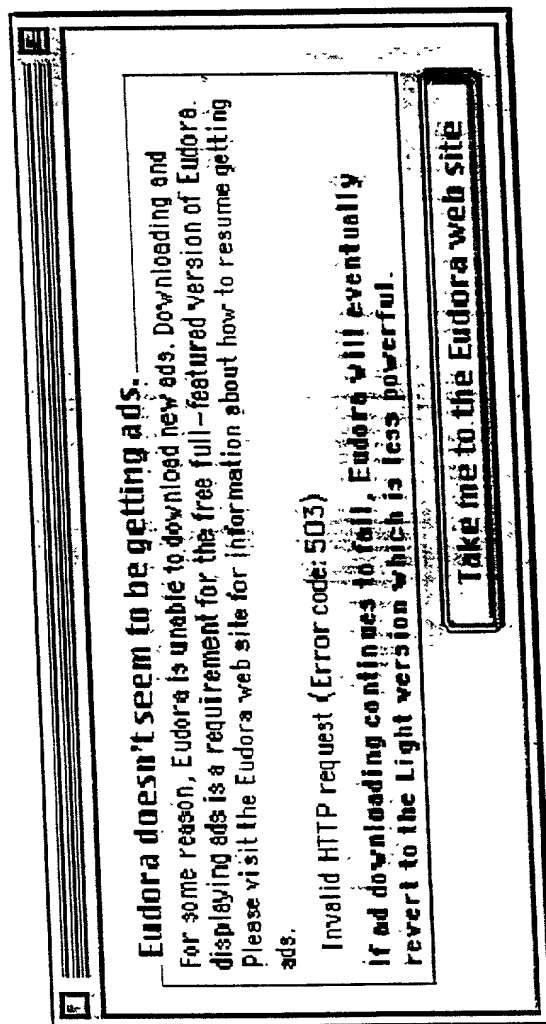
Fig. 15G

```

////////////////////////////////////
// Perform housekeeping when we put up an ad
AdStartBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// For regular ads
if ( it's a regular ad )
{
ad.thisShowTime = 0
ad.lastShownDate = now
}
}
// end AdStartBookkeeping

```

Fig. 15H



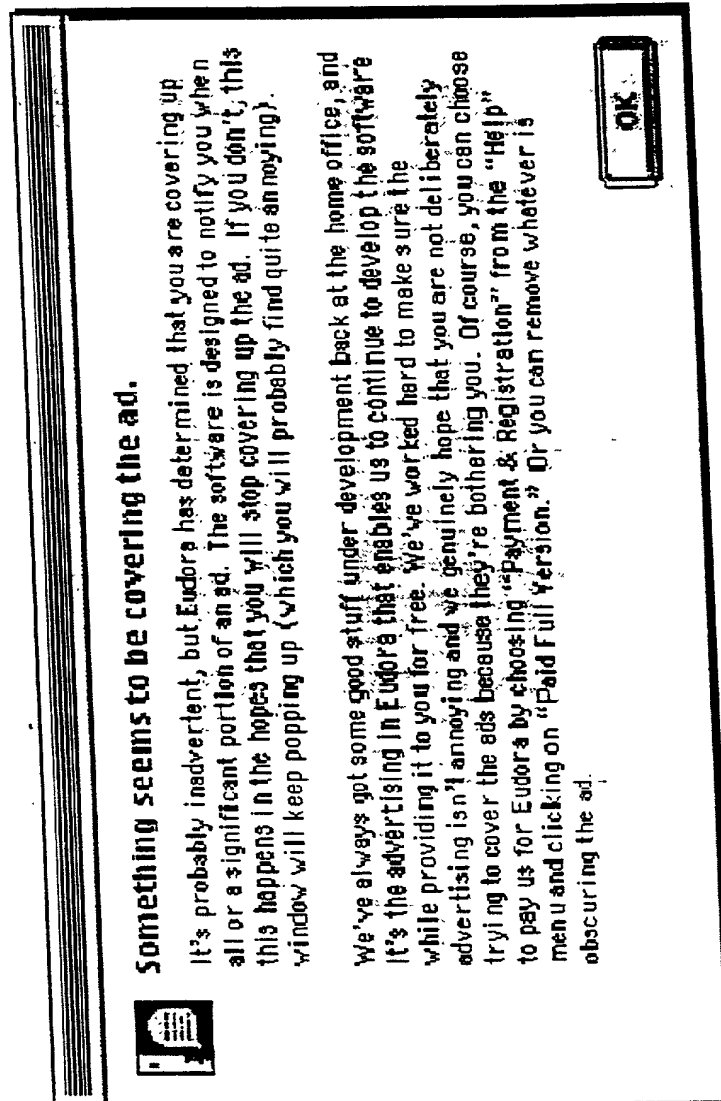


Fig. 17B

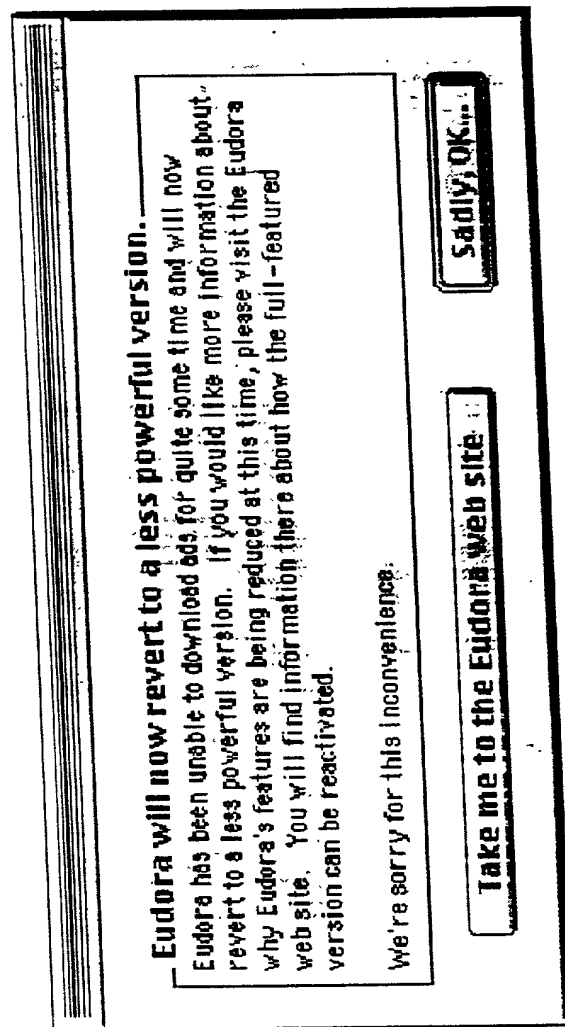


Fig. 17C

We'd like to know how you use Eudora.

In order to make Eudora work as well as possible, it's important that we know how people use it. We ask users for this information at random. Looks like it's your turn. If you're open to helping us this way, all you have to do is click "Generate Info" below and a message will be created. You can review the contents of the message if you like, and then send it to us or not -- that's up to you.

We value our privacy; we're pretty sure you value yours. So we want you to know what we'll be collecting and give you a chance to eliminate anything you don't want to send. Simply uncheck the boxes next to any information you'd rather not send.

Please understand that as soon as we receive your email, we will throw away the headers that identify the mail as coming from you. You see, we don't actually need to know who you are to find your information helpful. So we promise to protect your privacy and turn you into "just a number."

It's OK to transmit statistics regarding:

- ☒ Your demographic data
- ☒ Advertisement information
- ☒ Non-personal settings
- ☒ Your Net/Eudora usage
- ☒ Eudora features you use

Generate Info

Cancel

Fig. 18A

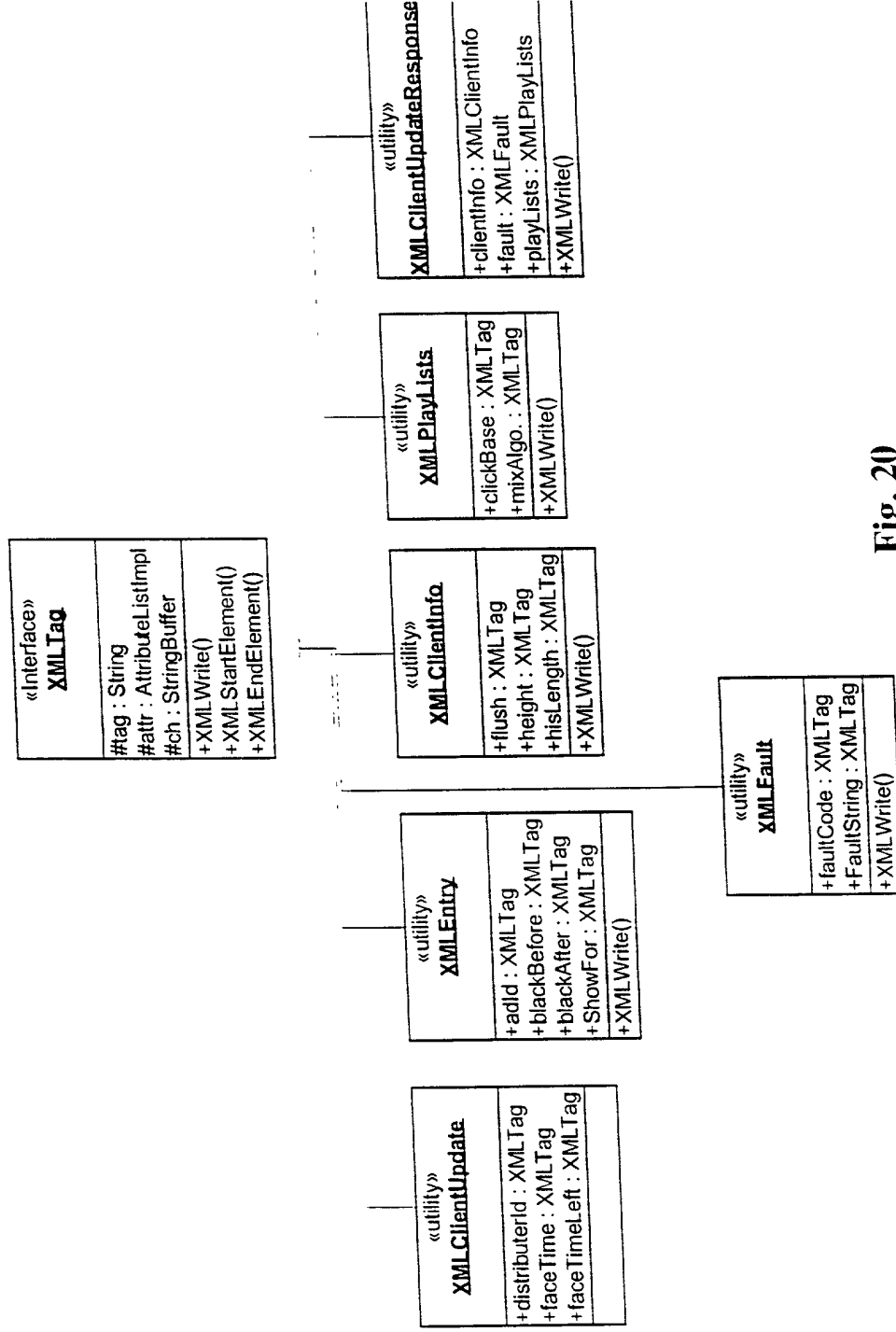


Fig. 20

8 The list of available ads advantageously can be built from the following query:

```
ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = 'I' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed ASC);
run out ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND AdType = 'R' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed ASC);
```

8 The time required to deliver the ads advantageously can be calculated in the following manner.

```
face time left for today [seconds] = faceTime[today] - faceTimeUsedToday
```

(Comment: Face time left for today is the number of seconds the servlet can use to deliver special ads today.)

```
predict face time [seconds] = SUM( faceTime[tomorrow + 1], ... faceTime[tomorrow + reqInterval] )
```

(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)

```
goal show time left [seconds] = predict face time - faceTimeLeft
```

(Comment: Goal show time left is the number of seconds that the software provider needs to fill with ads.)

Fig. 21A

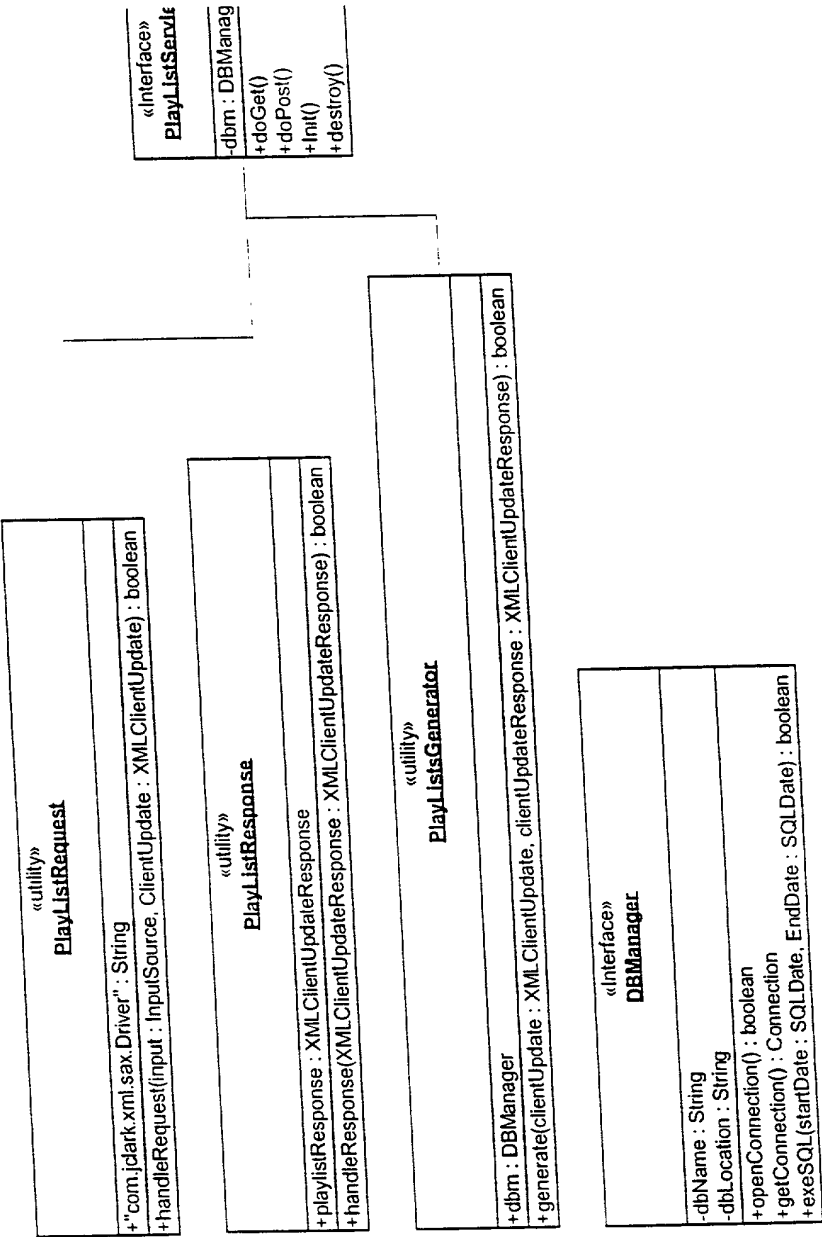


Fig. 22

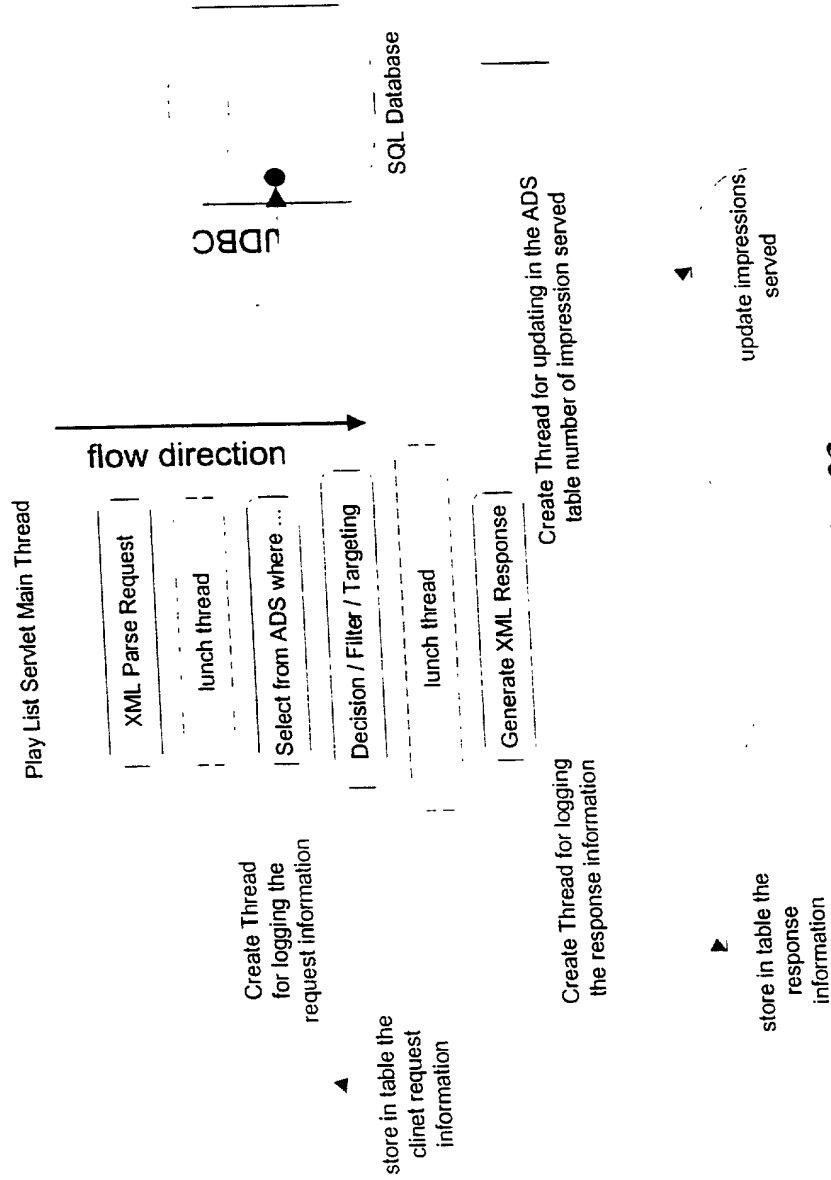


Fig. 23